Cite as: William.A.Hay: A low-Mach number solver for variable density flows. In Proceedings of CFD with OpenSource Software, 2018, Edited by Nilsson. H., http://dx.doi.org/10.17196/OS_CFD#YEAR_2018

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

A low-Mach number solver for variable density flows

Developed for OpenFOAM-v1806

Author:
William. A. Hay
Universitè catholique de Louvain

Peer reviewed by:
Silje Almeland
Håkan Nilsson

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

December 27, 2018

Learning outcomes

The reader will learn:

How to use

- The low-Mach number solver in a simple tutorial of a Rayleigh-Bénard convection cell

The theory of

- the low-Mach number approximation and its application to variable density flows with strong temperature gradients

How to implement

- the necessary changes to compile a true low-Mach solver from pre-existing solvers.

How to modify

- function objects to assess heat transfer in Rayleigh-Bénard convection

Prerequisites

In order to get maximum benefit out of this report the reader should :

- be able to run standard tutorials like heatTransfer/buoyantPimpleFoam/hotRoom
- have knowledge of natural convection and the role of the non-dimensional Rayleigh, Prandtl and Nusselt Numbers
- have an insight into the physics of turbulent Rayleigh-Bénard convection from [1]

Contents

Chapter 1

Theory

1.1 Introduction

Variable density low-Mach number flows appear frequently in nature and industrial processes with examples including atmospheric, oceanographic and combustion flows. A low-Mach number flow is defined as such when pressure variations are small, but temperature/concentration (and hence density) gradients are large.

In many numerical methods for simulations of flows with variable density the change in density is only captured in the gravitational term of the momentum equation and density is considered constant in the remaining unsteady and convection terms. This is the Boussinesq approximation, which Ferziger and Peric [2] suggest can be used for liquid flows if the temperature gradient is less than 2K and for gaseous flows if the temperature gradient is less than 15K. Using the Boussinesq approximation outside of this range can potentially produce qualitatively erroneous results.

If the Mach number in the entire flow field is low then the low-Mach number approximation can be applied to the governing equations. The result is a set of equations that can be solved via a predictor-corrector method, such as that of [3]. In such a projection method the divergence of the discretized momentum equation leads to a pressure Poisson equation. Unlike in constant density flows however, where the divergence of the velocity field is zero, in variable density flows the divergence of $\rho\mathbf{u}$ is required. Using the continuity equation we then enforce continuity by replacing this divergence term with the RHS of the continuity equation. This tutorial will outline how and why these changes can be implemented within the structure of an existing OpenFOAM solver.

1.2 Governing Equations

The aim of this section is to provide the governing equations to be solved by the low-Mach number variable density solver. First, the compressible Navier-Stokes equations are written in their entirety and the low-Mach number approximation is carried out, enabling the full low-Mach number system of equations to be written.

1.2.1 Continuity

The continuity equation is given by

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \mathbf{u}) = 0 \tag{1.1}$$

Where ρ ($\frac{\mathrm{kg}}{\mathrm{m}^3}$) is the fluid density and $\mathbf{u}$ the velocity ($\frac{\mathrm{m}}{\mathrm{s}}$).

1.2.2 Momentum Equation

Assuming the only relevant body force to be that of buoyancy the momentum-balance law equation is given by

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{u}\mathbf{u}) = -\nabla p + \nabla\cdot\tau + \rho\mathbf{g} \tag{1.2}$$

Where p is the pressure (Pa), $\mathbf{g}$ is the constant of gravitational acceleration ($9.81\,\frac{\mathrm{m}}{\mathrm{s}^2}$) and τ is the viscous stress tensor, expanded as follows

$$\tau = \mu\Big(\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^{\mathrm{T}}\right) - \frac{2}{3}(\nabla\cdot\mathbf{u})\,\mathbf{I}\Big)$$

Where μ ($\frac{\mathrm{kg}}{\mathrm{m\,s}}$) is the fluid viscosity and $\mathbf{I}$ is the identity matrix. It is noted that the bulk viscosity is considered zero (everywhere) in OpenFOAM and the second coefficient of viscosity is estimated as $\frac{2}{3}\mu$. Finally, the term containing the velocity divergence is non-zero, unlike the incompressible case.

The buoyancy and the pressure gradient terms are grouped together to improve the robustness of the solver. The hydrodynamic pressure $p' = p - \rho\mathbf{g}\cdot\mathbf{h}$ is used in the gradient term and a new source term appears on the RHS.

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{u}\mathbf{u}) = -\nabla p' + \nabla\cdot\tau - \mathbf{g}\cdot\mathbf{h}\nabla\rho \tag{1.3}$$

1.2.3 Energy Equation

The energy equation is written in terms of specific enthalpy with the viscous terms considered negligible, an assumption which stands for low-Mach number flows. Assuming no heat source/sink within the material volume and that the material derivative of pressure is written as $\frac{\mathrm{d}p}{\mathrm{d}t}$, the energy-balance law is given by

$$\frac{\partial(\rho h)}{\partial t} + \nabla\cdot(\rho\mathbf{u}h) = \nabla\cdot(k\nabla T) + \frac{\mathrm{d}p}{\mathrm{d}t} \tag{1.4}$$

Where h ($\frac{\mathrm{kJ}}{\mathrm{kg}}$) is the specific enthalpy, the heat flux divergence term has been expanded in accordance with Fourier's law; k is the thermal conductivity $\left(\frac{\mathrm{W}}{\mathrm{m\,K}}\right)$ and T is the temperature (K).

1.2.4 Equation of State

In this tutorial the fluid is treated as a perfect gas

$$p = \rho RT = \frac{\rho T}{\gamma M^2} \tag{1.5}$$

Where R is the gas constant, γ is the ratio of specific heats at a reference temperature $\left(\frac{C_p}{C_v}\right)$ and M is the Mach number, which for low-Mach number flows is considered small.

1.3 Low-Mach Number Approximation

For a full derivation of the non-dimensional low-Mach number equations see [4], below are the important aspects required for understanding of this tutorial. The low-Mach number equations are found by expanding the flow variables; ρ, $\mathbf{u}$, T and p, as a power series around $\epsilon = \gamma M^2$.

$$\rho = \rho^0 + \epsilon\rho^1 + \mathcal{O}\left(\epsilon^2\right)$$
$$\mathbf{u} = \mathbf{u}^0 + \epsilon\mathbf{u}^1 + \mathcal{O}\left(\epsilon^2\right)$$

$$T = T^0 + \epsilon T^1 + \mathcal{O}\left(\epsilon^2\right)$$

$$p = \frac{\rho T}{\gamma M^2} = \frac{1}{\epsilon} p^0 + p^1 + \mathcal{O}\left(\epsilon\right)$$

Where we now see that the pressure has been decomposed into two contributions; the thermodynamic pressure, p^0 and the dynamic pressure, p^1 hereon referred to as p_d. The consequence of this is two-fold, first knowing that $\epsilon = \gamma M^2 << 1$, the pressure appearing in the material derivative term of the energy equation is predominantly the thermodynamic term, p^0. Secondly, the spatially uniform nature of p^0 means that the pressure appearing in the gradient term of the momentum equation is uniquely the dynamic term, p_d. The final set of governing equations with all superscript values dropped (except in the pressure terms) is thus given as follows

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \mathbf{u}) = 0 \tag{1.6}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p_d' + \nabla \cdot \tau + \mathbf{g} \cdot \mathbf{h} \nabla \rho \tag{1.7}$$

Where $p_d' = p_d - \rho \mathbf{g} \cdot \mathbf{h}$.

$$\frac{\partial (\rho h)}{\partial t} + \nabla \cdot (\rho \mathbf{u} h) = \nabla \cdot (k \nabla T) + \frac{\partial p^0}{\partial t} \tag{1.8}$$

With the following equation of state to close the system :

$$\rho = \frac{p^0}{RT} \tag{1.9}$$

The only contribution to the pressure material derivative in the energy equation is the partial time derivative as, as already stated, the p^0 term is constant in space but, under certain conditions only, not in time. If the compuatational domain is open to atmosphere then the thermodynamic pressure is constant (in space *and* time) and equal to that of atmospheric pressure. In a closed domain however p^0 can vary in time, although the total mass must remain constant and equal to the volume integral of the density [4]. The p^0 term is then calculated as follows

$$p^0 = \frac{M_0}{\int \frac{1}{RT} \mathrm{dV}} = \frac{M_0}{\int \psi \mathrm{dV}} \tag{1.10}$$

Where the total mass, must be constant in time and equal to M_0, a check which is later carried out in the code. For the case where the ideal gas law is the equation of state, $\psi \left(\frac{\mathrm{m}^2}{\mathrm{s}^2}\right)$, the compressibility, is set to $\frac{1}{RT}$.

1.4 Numerical Algorithm

In this section Chapter 3.8 from [5] is expanded and the discretisation procedure of the Navier-Stokes system for variable density flows is explained. In his PhD, Jasak refers the reader to [6] for an understanding of the fully compressible PISO algorithm but the case for low-Mach number PISO algorithm has no clear reference, a fact which motivates this tutorial.

The changes outlined in the introduction must be introduced, but the idea remains the same as in the constant density PISO algorithm; use the momentum equations to provide the velocity field and a combination of the continuity and the momentum equations to formulate a pressure Poisson equation.

1.4.1 Variable Density Pressure Poisson Equation

A semi-discretized version of the momentum equation (source terms in differential form) is formulated as follows:

$$a_p U_p + \sum_N a_N U_N - \frac{U^0}{\Delta t} = -\nabla p'_d - \mathbf{g} \cdot \mathbf{h} \nabla \rho \tag{1.11}$$

From which we can get a relation for the velocity at point P as follows

$$U_p = \frac{1}{a_p} H(U) - \frac{1}{a_p} \nabla p'_d - \frac{1}{a_p} \mathbf{g} \cdot \mathbf{h} \nabla \rho \tag{1.12}$$

Where $H(U) = -\sum_N a_N U_N + \frac{U^0}{\Delta t}$

Multiplying equation (1.12) through by ρ gives the term inside the divergence operator of the continuity equation (1.6) as follows

$$\rho U_p = \frac{\rho}{a_p} H(U) - \frac{\rho}{a_p} \nabla p'_d - \frac{\rho}{a_p} \mathbf{g} \cdot \mathbf{h} \nabla \rho \tag{1.13}$$

In the spirit of Rhie-Chow the pressure equation is solved at the cell faces as opposed to the cell centres. The ρU values on the cell faces are thus found by interpolating to the face, denoted here by the notation $(...)_f$. Equation (1.14) is later used to determine ϕ, the face flux, which for variable density flows is equal to $(\rho \mathbf{U})_f \cdot \mathbf{S}_f$. Incidentally, the fvc::flux(...) OpenFOAM function does exactly this procedure; first interpolate to the face, then take the dot product with the face normal.

$$F = \left(\rho U\right)_f \cdot \mathbf{S}_f = \left(\frac{\rho}{a_p} H(U)\right)_f \cdot \mathbf{S}_f - \left(\frac{\rho}{a_p}\right)_f (\mathbf{g} \cdot \mathbf{h})_f \left(\nabla \rho\right)_f \cdot \mathbf{S}_f - \left(\frac{\rho}{a_p}\right)_f \left(\nabla p'_d\right)_f \cdot \mathbf{S}_f \tag{1.14}$$

Taking the divergence of Equation (1.13) gives the following

$$\nabla \cdot (\rho U_p) = \nabla \cdot \left(\frac{\rho}{a_p} H(U) - \frac{\rho}{a_p} \mathbf{g} \cdot \mathbf{h} \nabla \rho\right) - \nabla \cdot \left(\frac{\rho}{a_p} \nabla p'_d\right) \tag{1.15}$$

Combining (1.15) and (1.6) gives the pressure Poisson equation as follows

$$\nabla \cdot \left(\frac{\rho}{a_p} \nabla p'_d\right) = \nabla \cdot \left(\frac{\rho}{a_p} H(U) - \frac{\rho}{a_p} \mathbf{g} \cdot \mathbf{h} \nabla \rho\right) + \frac{\partial \rho}{\partial t} \tag{1.16}$$

In the third term on the RHS of (1.16) ρ can be replaced by ψp^0 to give the final discretized form of the variable density pressure poisson equation as follows

$$\underbrace{\sum_f \left(\left(\frac{\rho}{a_p}\right)_f (\nabla p'_d)_f\right) \cdot \mathbf{S}_f}_{fvm::laplacian(rhorAUf,pd)} = \underbrace{\sum_f \left(\left(\frac{\rho}{a_p} H(U)\right)_f - \left(\frac{\rho}{a_p}\right)_f (\mathbf{g} \cdot \mathbf{h})_f \left(\nabla \rho\right)_f\right) \cdot \mathbf{S}_f}_{fvc::div(phiHbyA)} + \underbrace{\frac{\partial \psi}{\partial t} p^0 \cdot V}_{fvc::ddt(psi)*p} \tag{1.17}$$

1.4.2 Transient Variable Density PISO Algorithm

The aim of the PISO algorithm is to use a segregated approach to solve the discretized Navier-Stokes system just described. Starting with known ρ, T, $\mathbf{u}$ and p at time n the full algorithm is then as follows

1. Using ρ^n and p_d^n solve for $\mathbf{u}^*$ (momentum predictor step)
2. Using ρ^n and $\mathbf{u}^*$ solve for h^* from energy equation (get T^*)
3. Using T^* find ψ^* from equation of state
4. Using integral of ψ^* find p^{0*} from (1.10)
5. Using p^{0*} and T^* find ρ^* from equation of state
6. Using $\mathbf{u}^*$ and ρ^* formulate the $H(U)^*$ operator
7. Using $H(U)^*$, and ρ^* formulate the pressure equation from (1.17)
8. Solve for p_d^* (pressure solution step)
9. Solve for $\mathbf{u}^{**}$ from (1.12) (explicit velocity correction)

The explicit nature of step 9 means that only the correction due to the pressure gradient term is taken into account here. That is to say that the correction due to the density related terms in (1.12) are ignored, likewise the contribution from the H(U) term. In the current algorithm, the PISO corrector step iterates over step 6-9 a certain number of times (defined by the user in fvSolution nCorrectors) to update the $H(U)^*$ term based on the updated velocity ($\mathbf{u}^{**}$). After a certain number of nCorrector iterations one has p_d^{n+1} and $\mathbf{u}^{n+1}$. It is noted that energy and momentum balance laws are coupled. In such a situation, [3] states that there should also be a corrector step for the temperature (and hence the density terms) which is not currently the case and this improvement task is left to the user if they wish to further improve the algorithm.

Chapter 2

Tutorial A : Build of lowMachBuoyantPimpleFoam

As already mentioned, the OpenFOAM solver which solves the set of equations most closely representing our problem is *buoyantPimpleFoam*; a transient solver for buoyant, turbulent flow of compressible fluids for ventilation and heat-transfer. Changes are predominantly required to the energy equation (EEqn.H), momentum equation (UEqn.H) and the PISO algorithm (pEqn.H); the latter requiring significant updates and resembling somewhat the pEqn.H in Reference [7]. Finally, we introduce new variables to the solver (via an additional file readFlowProperties.H) to help save time in post-processing for the specific case of Rayleigh-Bénard convection in a *cube*. All source code associated with the moving reference frame (MRF) is removed for clarity.

2.1 Copying and Renaming the Solver

In this first section the *buoyantPimpleFoam* solver is copied, renamed and recompiled.

```
foam
cp -r --parents applications/solvers/heatTransfer/buoyantPimpleFoam/ $WM_PROJECT_USER_DIR
ufoam
cd applications/solvers/heatTransfer/
mv buoyantPimpleFoam/ lowMachBuoyantPimpleFoam
cd lowMachBuoyantPimpleFoam/
mv buoyantPimpleFoam.C lowMachBuoyantPimpleFoam.C
sed -i s/buoyantPimpleFoam/lowMachBuoyantPimpleFoam/g Make/files
wclean
wmake
```

At this point the user will have an identical solver to *buoyantPimpleFoam* with a different name.

2.2 Create readRayleighBenardNusselt.H

The following is required for post-processing of the Nusselt number in a cubic domain for Rayleigh-Bénard convection. We first add lines of code (lines 1 - 59) that searches for a dictionary called flowProperties containing the definition of the user-defined (and case specific) variables of TRef, deltaTRef, rhoRef, URef, RaRef and PrRef. Then we create the new fields (lines 60 - 143) that are required to dynamically calculate the Nusselt number introduced in 3.1.

Listing 2.1: readRayleighBenardNusselt.H

```
1
2  // For Rayleigh-Benard Nusselt number in a cube
3
4  Info<< "Reading flowProperties\n" << endl;
5
6  IOdictionary flowProperties
7  (
8      IOobject
9      (
10         "flowProperties",
11         runTime.constant(),
12         mesh,
13         IOobject::MUST_READ_IF_MODIFIED,
14         IOobject::NO_WRITE
15     )
16 );
17
18 dimensionedScalar TRef
19 (
20     "TRef",
21     dimTemperature,
22     flowProperties
23 );
24
25 dimensionedScalar deltaTRef
26 (
27     "deltaTRef",
28     dimTemperature,
29     flowProperties
30 );
31
32 dimensionedScalar rhoRef
33 (
34     "rhoRef",
35     dimDensity,
36     flowProperties
37 );
38
39 dimensionedScalar URef
40 (
41     "URef",
42     dimVelocity,
43     flowProperties
44 );
45
46
47 dimensionedScalar RaRef
48 (
49     "RaRef",
50     dimless,
51     flowProperties
52 );
53
54 dimensionedScalar PrRef
```

```
55 (
56     "PrRef",
57     dimless,
58     flowProperties
59 );
60
61 Info<< "Creating field nonDimT\n" << endl;
62 volScalarField nonDimT
63 (
64     IOobject
65     (
66         "nonDimT",
67         runTime.timeName(),
68         mesh,
69         IOobject::MUST_READ,
70         IOobject::AUTO_WRITE
71     ),
72     mesh
73 );
74
75 Info<< "Creating field nonDimRho\n" << endl;
76 volScalarField nonDimRho
77 (
78     IOobject
79     (
80         "nonDimRho",
81         runTime.timeName(),
82         mesh,
83         IOobject::MUST_READ,
84         IOobject::AUTO_WRITE
85     ),
86     mesh
87 );
88
89 Info<< "\nReading field nonDimU\n" << endl;
90 volVectorField nonDimU
91 (
92     IOobject
93     (
94         "nonDimU",
95         runTime.timeName(),
96         mesh,
97         IOobject::MUST_READ,
98         IOobject::AUTO_WRITE
99     ),
100    mesh
101 );
102
103 Info<< "Creating field nusseltOne\n" << endl;
104 volScalarField nusseltOne
105 (
106     IOobject
107     (
108         "nusseltOne",
```

```
109         runTime.timeName(),
110         mesh,
111         IOobject::MUST_READ,
112         IOobject::AUTO_WRITE
113     ),
114     mesh
115 );
116 
117 Info<< "Creating field nusseltTwo\n" << endl;
118 volScalarField nusseltTwo
119 (
120     IOobject
121     (
122         "nusseltTwo",
123         runTime.timeName(),
124         mesh,
125         IOobject::MUST_READ,
126         IOobject::AUTO_WRITE
127     ),
128     mesh
129 );
130 
131 Info<< "Creating field nusseltTotal\n" << endl;
132 volScalarField nusseltTotal
133 (
134     IOobject
135     (
136         "nusseltTotal",
137         runTime.timeName(),
138         mesh,
139         IOobject::MUST_READ,
140         IOobject::AUTO_WRITE
141     ),
142     mesh
143 );
```

2.3 Create calculateRayleighBenardNusselt.H

The following is required for post-processing of the Nusselt number in a cubic domain for Rayleigh-Bénard convection. Using the declared fields from readRayleighBenardNusselt.H we now *define* them (lines 4-6 and 32-34) and finally create the remaining fields required to dynamically calculate the Nusselt number (lines 8-30).

Listing 2.2: calculateRayleighBenardNusselt.H

```
1 
2 // To calculate the Nusselt number in RBC in a cube
3 
4 nonDimU = (U/URef);
5 nonDimRho = (rho/rhoRef);
6 nonDimT = (T-TRef)/(deltaTRef);
7 
8 volVectorField gradNonDimT(fvc::grad(nonDimT));
9 
```

```
10 volScalarField gradNonDimTy
11 (
12     IOobject
13     (
14         "gradNonDimTy",
15         runTime.timeName(),
16         mesh
17     ),
18     gradNonDimT.component(vector::Y)
19 );
20
21 volScalarField nonDimUy
22 (
23     IOobject
24     (
25         "nonDimUy",
26         runTime.timeName(),
27         mesh
28     ),
29     nonDimU.component(vector::Y)
30 );
31
32 nusseltOne = sqrt(RaRef*PrRef)*nonDimUy*nonDimRho*nonDimT;
33 nusseltTwo = hRef*gradNonDimTy; // hRef is to nonDimensionalize the denominator
34 nusseltTotal = nusseltOne - nusseltTwo;
```

2.4 Modify createFieldRefs.H

In the createFieldRefs.H file from *buoyantPimpleFoam* the compressibility field, psi, is defined and declared. In the low-Mach number solver psi is no longer a reference field, i.e. it is used in the calculation of other fields in the solver, it is thus moved to createFields.H and the createFieldRefs.H file is as follows

Listing 2.3: createFieldRefs.H

```
1 const volScalarField& T = thermo.T();
```

2.5 Modify createFields.H

The hydrodynamic pressure field, p'_d, is defined and declared in createFields.H as pd (line 55 - 67), note that the ' has been dropped. The psi field is defined and declared (lines 624 - 636). Its value is stored from the previous time-step thus allowing for the calculation of the $\frac{\partial \psi}{\partial t}$ term later on (see pEqn.H lines 63 - 66). The thermodynamic pressure, p^0, is defined as p (lines 34 - 37), it is declared in accordance with Equation (1.10), which itself requires a definition of M_0, given in line 622 as the dimensionedScalar totalMass0.

Listing 2.4: createFields.H

```
1 Info<< "Reading thermophysical properties\n" << endl;
2
3 autoPtr<rhoThermo> pThermo(rhoThermo::New(mesh));
4 rhoThermo& thermo = pThermo();
5 thermo.validate(args.executable(), "h");
```

```
6
7   volScalarField rho
8   (
9       IOobject
10      (
11          "rho",
12          runTime.timeName(),
13          mesh,
14          IOobject::MUST_READ,
15          IOobject::AUTO_WRITE
16      ),
17      thermo.rho()
18  );
19
20  dimensionedScalar totalMass0 = fvc::domainIntegrate(rho); // total mass based on thermo.rho AT t0
21
22  volScalarField psi
23  (
24      IOobject
25      (
26          "psi",
27          runTime.timeName(),
28          mesh
29      ),
30      thermo.psi()
31  );
32  psi.oldTime(); // need to store for ddt term
33
34  volScalarField& p = thermo.p();
35
36  p = totalMass0/fvc::domainIntegrate(psi);
37  Info<< "min/max(p) = " << min(p).value() << ", " << max(p).value() << endl;
38
39  Info<< "Reading field U\n" << endl;
40  volVectorField U
41  (
42      IOobject
43      (
44          "U",
45          runTime.timeName(),
46          mesh,
47          IOobject::MUST_READ,
48          IOobject::AUTO_WRITE
49      ),
50      mesh
51  );
52
53  #include "compressibleCreatePhi.H"
54
55  Info<< "Creating field pd\n" << endl;
56  volScalarField pd
57  (
58      IOobject
59      (
```

```
60          "pd",
61          runTime.timeName(),
62          mesh,
63          IOobject::MUST_READ,
64          IOobject::AUTO_WRITE
65      ),
66      mesh
67  );
68
69  Info << "Creating turbulence model.\n" << nl;
70  autoPtr<compressible::turbulenceModel> turbulence
71  (
72      compressible::turbulenceModel::New
73      (
74          rho,
75          U,
76          phi,
77          thermo
78      )
79  );
80
81  #include "readGravitationalAcceleration.H"
82  #include "readhRef.H"
83  #include "gh.H"
84
85
86  Info<< "Creating field dpdt\n" << endl;
87  volScalarField dpdt
88  (
89      IOobject
90      (
91          "dpdt",
92          runTime.timeName(),
93          mesh
94      ),
95      mesh,
96      dimensionedScalar("dpdt", p.dimensions()/dimTime, 0)
97  );
98
99  mesh.setFluxRequired(pd.name());
100
101 #include "createFvOptions.H"
```

2.6 Modify EEqn.H

The energy equation is in specific enthalpy form, this must be respected in the runCaseName/constant/thermophysicalProperties choice made by the user. The $\frac{\partial p}{\partial t}$ term has been included (line 7) in accordance with Equation (1.8). In the heat transfer solvers of OpenFOAM, the temperature in Fourier's law is replaced using the relation $\mathrm{T} = \frac{h}{C_p}$, the resulting $\frac{k}{C_p}$ prefactor is then encorporated into the alphaEff() object (line 8) which is associated with the chosen turbulence model. This is ofcourse only possible if C_p is considered constant and the user should be wary of the physical limitations of this computational method.

Listing 2.5: EEqn.H

```
1  {
2      volScalarField& h = thermo.he();
3
4      fvScalarMatrix EEqn
5      (
6          fvm::ddt(rho, h) + fvm::div(phi, h)
7        - dpdt
8        - fvm::laplacian(turbulence->alphaEff(), h)
9       ==
10       fvOptions(rho, h)
11     );
12
13     EEqn.relax();
14
15     fvOptions.constrain(EEqn);
16
17     EEqn.solve();
18
19     fvOptions.correct(h);
20
21     thermo.correct();
22
23     Info<< "min/max(T) = "
24         << min(T).value() << ", " << max(T).value() << endl;
25 }
```

2.7 Modify pEqn.H

In lines 2-4 the thermodynamic pressure is calculated for the new time-step. In lines 6-18 info statements will show whether or not mass is conserved as we march forward in time and further that the thermodynamic pressure is calculated correctly. The key step here is the inclusion of the fvc::ddt(psi)*p term in the pressure Poisson equation (line 42) to enforce continuity. Where *psi* is declared via the equation of state, selected by the user in constant/thermophysicalProperties. As a consequence of adding this term explicitly (see use of fvc::), the pdEqn fvScalarMatrix is ill-conditioned. We therefore add the term $\frac{\partial(\psi\, p_d')}{\partial t}$ implicitly (using fvm:: in line 41) and subtract explicitly (using fvc:: in line 42) to increase the diagonally dominance of the matrix and thus help with convergence. This numerical method was taken from Reference [8].

Listing 2.6: pEqn.H

```
1
2  psi = thermo.psi();
3  p = totalMass0/fvc::domainIntegrate(psi);
4  rho = thermo.rho();
5
6  dimensionedScalar totalMass = fvc::domainIntegrate(rho);
7  scalar constantMass = (totalMass0/totalMass).value();
8  Info<< "Should be constant in time " << constantMass << endl;
9
10
11     Info<< "min/max(rho) = "
12         << min(rho).value() << ", " << max(rho).value() << endl;
```

```
13
14      Info<< "min/max(psi) = "
15          << min(psi).value() << ", " << max(psi).value() << endl;
16
17      Info<< "min/max(p) = "
18          << min(p).value() << ", " << max(p).value() << endl;
19
20  volScalarField rAU(1.0/UEqn.A());
21  surfaceScalarField rhorAUf("rhorAUf", fvc::interpolate(rho*rAU));
22  volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
23  surfaceScalarField phig("phig", -rhorAUf*ghf*fvc::snGrad(rho)*mesh.magSf());
24
25  surfaceScalarField phiHbyA
26  (
27      "phiHbyA",
28      (
29          fvc::flux(rho*HbyA)
30      )
31      + phig
32  );
33
34  constrainPressure(pd, rho, U, phiHbyA, rhorAUf);
35
36  while (pimple.correctNonOrthogonal())
37  {
38      fvScalarMatrix pdEqn
39      (
40          fvm::ddt(psi, pd) // implicit addition of pd
41        - fvc::ddt(psi, pd) // explicit subtraction of pd
42        + fvc::ddt(psi)*p
43        + fvc::div(phiHbyA)
44        - fvm::laplacian(rhorAUf, pd)
45       ==
46          fvOptions(psi, pd, rho.name())
47      );
48
49      pdEqn.solve(mesh.solver(pd.select(pimple.finalInnerIter())));
50
51      if (pimple.finalNonOrthogonalIter())
52      {
53          phi = phiHbyA + pdEqn.flux();
54          U = HbyA + rAU*fvc::reconstruct((pdEqn.flux() + phig)/rhorAUf);
55          U.correctBoundaryConditions();
56          fvOptions.correct(U);
57      }
58  }
59
60  #include "rhoEqn.H"
61  #include "compressibleContinuityErrs.H"
62
63  if (thermo.dpdt())
64  {
65      dpdt = fvc::ddt(p);
66  }
```

```
67
68 #include "calculateRayleighBenardNusselt.H"
```

2.8 Modify UEqn.H

The pressure gradient term is the only change required in the UEqn.H file where pd is now used (line 33). In the momentum predictor step (which should be switched on by the user in runCaseName/system/fvSolution) Equation (1.7) is solved.

Listing 2.7: UEqn.H

```
1   fvVectorMatrix UEqn
2    (
3        fvm::ddt(rho, U)
4      + fvm::div(phi, U)
5      + turbulence->divDevRhoReff(U)
6     ==
7      fvOptions(rho, U)
8    );
9
10   UEqn.relax();
11
12   fvOptions.constrain(UEqn);
13
14   if (pimple.momentumPredictor())
15   {
16       solve
17       (
18           UEqn
19        ==
20           fvc::reconstruct
21           (
22              (
23                  - ghf*fvc::snGrad(rho)
24                  -fvc::snGrad(pd)
25              )*mesh.magSf()
26           )
27       );
28
29       fvOptions.correct(U);
30   }
```

2.9 Compiling the Solver

No extra libraries are required and the code can be compiled with the following

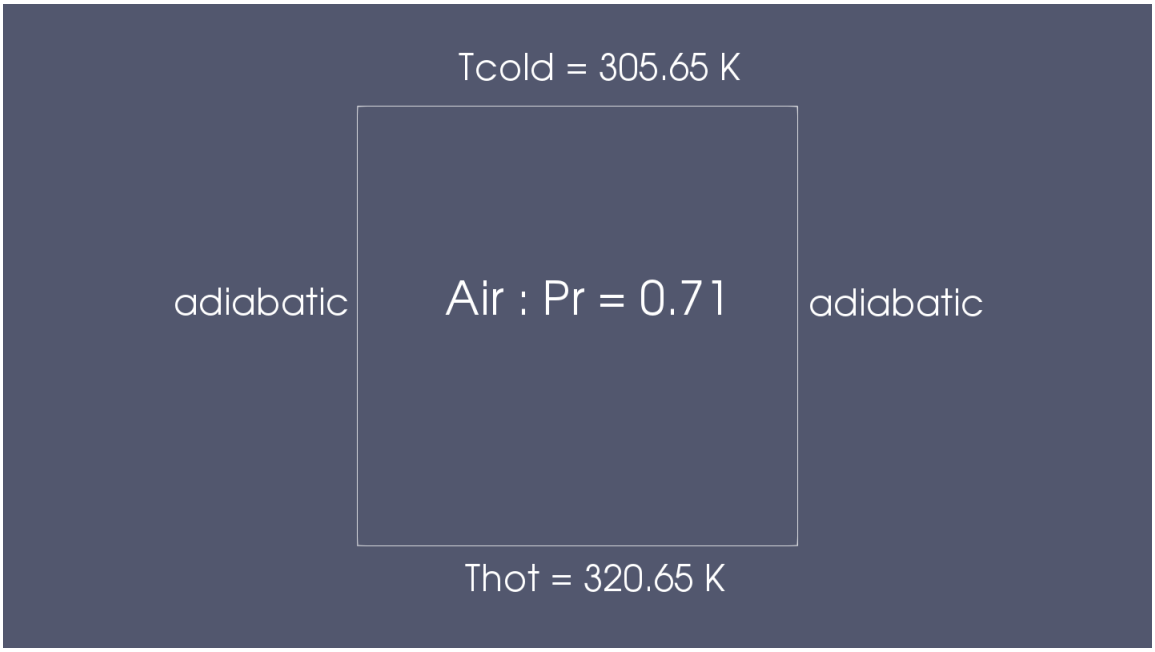
```
ufoam
cd applications/solvers/heatTransfer/lowMachBuoyantPimpleFoam
wclean
wmake
```

At this point the user will have a low-Mach-number solver with additional post-processing capabilities for Rayleigh-Bénard convection in a cube. If required, these additional capabilities can be switched off by deleting the two *RayleighBenardNusselt.H files, removing the appropriate lines of code in lowMachBuoyantPimpleFoam.C (line 58) and pEqn.H (line 68) and then recompiling.

Chapter 3

Tutorial B: A Rayleigh-Bénard Convection Test Case

As a simple tutorial, we will carry out a simulation of Rayleigh-Bénard convection in a 2D square domain with adiabatic sidewalls at a Rayleigh number of 10^5 using air as the working fluid, hence a Prandtl number of 0.7.



3.1 Background

Rayleigh-Bénard Convection is a classical problem in fluid mechanics which can be summarized as a fluid filled container heated from below and cooled from above. The system is determined by the two dimensionless control parameters of Rayleigh number and Prandtl number [1].

$$Ra = \frac{g\beta\Delta T L^3}{\nu\alpha} \qquad Pr = \frac{\nu}{\alpha}$$

Where β is the coefficient of thermal expansion ($\frac{1}{K}$), ΔT is the temperature difference (K) between the upper and lower walls, L is the distance between the same walls (m), ν is the kinematic viscosity ($\frac{m^2}{s}$) and α is the thermal diffusivity ($\frac{m^2}{s}$).

The system response to the imposed Ra and Pr can be measured in terms of a dimensionless heat flux (Nusselt number) across the geometry. For the case of a cube, this is given either by a time and area averaged slice taken at a chosen height in the domain [9]

$$Nu_y = \underbrace{\sqrt{RaPr}\,\langle \rho v T \rangle_{xzt}}_{component\,1} - \underbrace{\frac{\partial \langle T \rangle_{xzt}}{\partial y}}_{component\,2} \tag{3.1}$$

Or alternatively by a time and volume average over the entire domain [9]

$$Nu = \sqrt{RaPr}\,\langle \rho v T \rangle_{xyzt} + 1 \tag{3.2}$$

At a Rayleigh number of 10^5 the flow is near the transition to turbulence so the simulation must be run until a statistically steady state is reached, at which point thermal (Nusselt number) and flow (Reynolds number) dynamics can be statisically assessed. We focus on Nusselt number in this tutorial where we carry out a Direct Numerical Simulation (DNS), i.e. no turbulence model is used. It is noted that a more interesting test case would be to increase the Rayleigh to a more turbulent (and hence) chaotic case (at least 10^6) - in such a study the time averaging function objects to be used would be more relevant. If the user wishes to do the simplest way to increase the Rayleigh number is to increase the height of the domain, it must then be checked that all Kolmogorov and Batchelor scales are captured with the new grid. References [10] and [11] provide more detail on for bulk and boundary layer refinement criteria.

Finally, the solution is considered to be statistically steady once the Nusselt number is constant in the cube when measured at different locations. This is achieved after a certain number of LSC free-fall times, involving a certain amount of trial and error. A LSC free-fall time is, approximately, the physical time taken for a fluid particle to circulate the domain and is hence calculated as the height of the domain (L) divided by the free fall velocity $\mathbf{u}_f = \sqrt{g\beta\Delta T L}$, in the simulations of this tutorial we consider that the solution is statistically steady after 500 LSC free-fall times and we then take statistics over 500 more.

3.2 Copying and Renaming Tutorial

The hotRoom tutorial provides the appropriate names for the boundary conditions that we wish to use. It is therefore copied to the users run directory where the Allrun and Allclean scripts are updated.

```
run
cp -r $FOAM_TUTORIALS/heatTransfer/buoyantPimpleFoam/hotRoom ./rayleighBenard2DTestCase
cd rayleighBenard2DTestCase
mv 0.orig 0
sed -i s/cleanCase0/cleanCase/g Allclean
cp $FOAM_TUTORIALS/heatTransfer/buoyantPimpleFoam/thermocoupleTestCase/Allrun .
```

3.3 Initial and Boundary Conditions

The fields associated with the turbulence are removed and the dynamic pressure field is created.

```
cd 0/
rm alphat epsilon k nut
mv p_rgh pd
sed -i s/"internalField uniform 1e5"/"internalField uniform 0"/g pd
sed -i s/"uniform 1e5"/"\$internalField"/g pd
cd ..
```

Check the pd file to see if the sed commands have functioned correctly, if not, make the changes manually in a text editor.

3.3.1 Modify T

```
vim 0/T
```

Ideally the initial temperature condition would be a linear temperature profile between the hot and the cold walls. Here, for simpicity, the internal field is initialized to T_{mean}. Further, the boundary condition for the frontAndBack patch should be empty **for all fields** as shown in the T example below

Listing 3.1: T

```
/*--------------------------------*- C++ -*----------------------------------*/
FoamFile
{
    version 2.0;
    format ascii;
    class volScalarField;
    location "0";
    object T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions [0 0 0 1 0 0 0];


internalField uniform 313.15;

boundaryField
{
    floor
    {
        type fixedValue;
        value uniform 320.65;
    }
    ceiling
    {
        type fixedValue;
        value uniform 305.65;
    }
    fixedWalls
    {
```

```
        type zeroGradient;
    }
    frontAndBack
    {
        type empty;
    }
}

// ************************************************************************* //
```

3.3.2 Create dummy variables for post-processing

The non-dimensional dummy variables are required to calculate the Nusselt number during the calculation.

```
cp 0/U 0/nonDimU
cp 0/nonDimU 0/nusseltOne
cp 0/nonDimU 0/nusseltTwo
cp 0/nonDimU 0/nusseltTotal
sed -i s/"uniform (0 0 0)"/"uniform 0"/g 0/nusselt*
sed -i s/"noSlip"/"zeroGradient"/g 0/nusselt*
sed -i s/volVector/volScalar/g 0/n*
sed -i s/volScalar/volVector/g 0/nonDimU
cp 0/nusseltOne 0/nonDimRho
cp 0/T 0/nonDimT
sed -i s/"uniform 320.65"/"uniform 0.5"/g 0/nonDimT
sed -i s/"uniform 305.65"/"uniform -0.5"/g 0/nonDimT
sed -i s/"uniform 313.15"/"uniform 0"/g 0/nonDimT
```

The dimensions in all the non-dimensional files (that is nonDimRho, nonDimT, nonDimU, nusseltOne, nusseltTotal and nusseltTwo) need to be dimensionless. That means editing line 18 in all files to be as follows

```
dimensions [0 0 0 0 0 0 0];
```

3.4 Thermophysical And Flow Properties

There is no turbulence model in a Direct Numerical Simulation, therefore the simulationType is set to laminar. For clarity this is not an indication of a laminar flow, but an indication of the absence of subgrid scale modelling.

```
sed -i s/"RAS"/"laminar"/g constant/turbulenceProperties
```

The equation of state is already set to perfectGas so no changes are required to the thermophysicalProperties file.

3.4.1 Create flowProperties

Create and open the flowProperties file as follows

```
vim constant/flowProperties
```

The following is required for the post-processing of the Rayleigh-Bénard convection in a cube. The purpose of the file is to set reference values for the specific case, these values are then used to dynamically calculate the Nusselt number as outlined in (2.2) and (2.3).

Listing 3.2: flowProperties

```
/*--------------------------------*- C++ -*----------------------------------*/
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    object flowProperties;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Reference temperature
TRef 313.15;

// Reference temperature
deltaTRef 15;

// Reference density
rhoRef  1.1272;

// Reference freefall velocity (beta*deltaT*g*H)^0.5
URef  0.1441;

// Reference Rayleigh
RaRef  1e5;

// Reference Pr
PrRef  0.712;
// ************************************************************************* //
```

3.4.2 Create hRef

A reference height is needed by the solver to non-dimensionalize the temperature gradient term used in the dynamic calculation of Nu, see line 33 of listing in (2.3). The template for the file is copied from a fireFoam tutorial as follows

```
cp $FOAM_TUTORIALS/combustion/fireFoam/LES/smallPoolFire3D/constant/hRef constant/
```

Lines 18 - 19 should then be updated to the following

Listing 3.3: hRef

```
dimensions [0 1 0 0 0 0 0];
```

```
value 0.0442;
```

3.5 Modify system/ files

As well as the usual files that require udpating (blockMeshDict, controlDict, fvSchemes, fvSolution) we introduce here some functionObjects that allow for dynamic calculation of the Nusselt number in the domain as well as some time averaging techniques for the velocity.

3.5.1 Modify blockMeshDict

Many changes are required to blockMeshDict, which is opened with the following command.

```
vim system/blockMeshDict
```

Ensure that the scale, vertice locations, mesh refinement and boundary names are updated.

Listing 3.4: blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*/
FoamFile
{
    version 2.0;
    format ascii;
    class dictionary;
    object blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

scale 0.0221;

vertices
(
    (-1 -1 -0.001)
    (1 -1 -0.001)
    (1 1 -0.001)
    (-1 1 -0.001)
    (-1 -1 0.001)
    (1 -1 0.001)
    (1 1 0.001)
    (-1 1 0.001)
);
blocks
(
hex (0 1 2 3 4 5 6 7) (32 32 1) simpleGrading (1 1 1)
);


boundary
(
    floor
    {
```

```
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
    ceiling
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }

    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);

edges
(
);

mergePatchPairs
(
);

// ************************************************************************* //
```

3.5.2 Modify finite volume files

In fvSchemes we introduce second-order time integration (backward) and divergence (Gauss linear) schemes.

```
sed -i s/"    default         Euler"/"    default         backward"/g
↪  system/fvSchemes
sed -i s/"    div(phi,U)      Gauss upwind"/"    div(phi,U)      Gauss linear"/g
↪  system/fvSchemes
sed -i s/"    div(phi,h)      Gauss upwind"/"    div(phi,h)      Gauss linear"/g
↪  system/fvSchemes
```

In fvSolution, 3 corrector steps are required in the PISO algorithm to reduce the time discretization errors to second order. See [12] and [13] for further details.

```
sed -i '/    pRefCell        0/d' system/fvSolution
sed -i '/    pRefValue       1e5/d' system/fvSolution
sed -i s/p_rgh/pd/g system/fvSolution
sed -i s/"    nCorrectors       2"/"    nCorrectors       3"/g system/fvSolution
```

Further, it is recommended that the tolerance for the pd field is reduced to at least 1e-9 and the relTol to 0.001.

```
    pd
    {
        solver GAMG;
        tolerance 1e-9;
        relTol 0.001;
        smoother GaussSeidel;
    };

    pdFinal
    {
        $pd;
        tolerance 1e-9;
        relTol 0;
    };
}
```

3.5.3 Modify controlDict

Many changes are required to controlDict, which is opened with the following command.

```
vim system/controlDict
```

Within controlDict we update the endTime, deltaT, writeControl, writeInterval and maxCo. We further add the line associated with sampleControls (line 46) and then add a series of functions that allow us to time and area-average the velocity and the Nusselt number in the cube (lines 48 - 110). First, the functionObject *fieldAverage*, time-averages the Nusselt number volScalarFields over a set time period (lines 50 - 87), then line 109 includes the *fieldTransfer* file, which transfers to volScalarFields to surfaces and thus handles the area-averaging. The volume-averaging of the time-averaged fields is carried out in lines 89 - 106.

Listing 3.5: controlDict

```
FoamFile
{
```

```
    version 2.0;
    format ascii;
    class dictionary;
    location "system";
    object controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application lowMachBuoyantPimpleFoam;

startFrom startTime;

startTime 0;

stopAt endTime;

endTime 60;

deltaT 1e-2;

writeControl adjustableRunTime;

writeInterval 5;

purgeWrite 0;

writeFormat ascii;

writePrecision 6;

writeCompression off;

timeFormat general;

timePrecision 6;

runTimeModifiable true;

adjustTimeStep yes;

maxCo 0.1;

#include "sampleControls"

functions
{
    fieldAverage1
    {
        type fieldAverage;
        libs ("libfieldFunctionObjects.so");
        timeStart 30;
        timeEnd 60;
        writeControl writeTime;
```

```
        fields
        (
            nonDimU
            {
                mean on;
                prime2Mean on; //RMS
                base time;
            }

            nusseltTotal
            {
                mean on;
                prime2Mean off;
                base time;
            }
            nusseltOne
            {
                mean on;
                prime2Mean off;
                base time;
            }
            nusseltTwo
            {
                mean on;
                prime2Mean off;
                base time;
            }

        );
    }

    volAverageNusseltField
    {
        type volFieldValue;
        libs ("libfieldFunctionObjects.so");
        log true;
        timeStart 30;
        timeEnd 60;
        writeControl writeTime;
        writeFields true;
        regionType all;
        operation volAverage;
        fields
        (
            nusseltOneMean
            nusseltTwoMean
            nusseltTotalMean
        );
    }

// #include "cuttingPlaneNusseltWalls"
    #include "fieldTransfer"
}
```

3.5.4 Copy sampling files

The sampleControl file is copied from another tutorial.

```
cp $FOAM_TUTORIALS/incompressible/simpleFoam/squareBend/system/sampleControls system/
```

The file should be edited to have a timeStart and timeEnd in line with the options selected in controlDict.

Listing 3.6: sampleControl

```
__surfaceFieldValue
{
    type surfaceFieldValue;
    libs ("libfieldFunctionObjects.so");
    timeStart 30;
    timeEnd 60;

    log on;
    enabled true;

    writeControl timeStep;
    writeInterval 1;

    writeFields false;
    surfaceFormat vtk;
    writeArea true;
}
```

3.5.5 Create fieldTransfer

The *fieldTransfer* file allows for the volFields, which were created as dummy variables in the simulation, to be transferred to a surfaceField. In this instance we interpolate the values of Nusselt number to slices made in the vertical axes of the domain.

```
vim fieldTransfer
```

Listing 3.7: fieldTransfer

```
// ************************************************************************* //

// Transcribe volume fields to surfaces.
fieldTransfer
{
    type surfMeshes;
    libs ("libsampling.so");
    log true;
    timeStart 30;
    timeEnd 60;
    writeControl none;
```

```
createOnRead true;
executeControl timeStep;
executeInterval 1;

fields (nusseltTotalMean nusseltOneMean nusseltTwoMean);

surfaces
(
    middle
    {
        type plane;
        source cells;
        planeType pointAndNormal;
        pointAndNormalDict
        {
            point (0 0 0);
            normal (0 1 0);
        }
        interpolate false;
    }
    top
    {
        type plane;
        source cells;
        planeType pointAndNormal;
        pointAndNormalDict
        {
            point (0 0.0116485 0); //2 X boundary layer
            normal (0 1 0);
        }
        interpolate false;
    }
    bottom
    {
        type plane;
        source cells;
        planeType pointAndNormal;
        pointAndNormalDict
        {
            point (0 -0.0116485 0);
            normal (0 1 0);
        }
        interpolate false;
    }
    veryTop
    {
        type plane;
        source cells;
        planeType pointAndNormal;
        pointAndNormalDict
        {
            point (0 0.0168742 0); // 1 X boundary layer
            normal (0 1 0);
        }
```

```
            interpolate false;
        }
        veryBottom
        {
            type plane;
            source cells;
            planeType pointAndNormal;
            pointAndNormalDict
            {
                point (0 -0.0168742 0);
                normal (0 1 0);
            }
            interpolate false;
        }
    );
}

areaAveragePlaneMiddle
{
    ${__surfaceFieldValue}

    regionType surface;
    name middle;

    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
areaAveragePlaneTop
{
    ${__surfaceFieldValue}

    regionType surface;
    name top;

    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
areaAveragePlaneBottom
{
    ${__surfaceFieldValue}

    regionType surface;
    name bottom;

    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
areaAveragePlaneVeryTop
{
    ${__surfaceFieldValue}

    regionType surface;
    name veryTop;
```

```
    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
areaAveragePlaneVeryBottom
{
    ${__surfaceFieldValue}

    regionType surface;
    name veryBottom;

    operation areaAverage;
    fields ( nusseltOneMean nusseltTwoMean nusseltTotalMean );
}
```

3.6 Run test case

From inside the case directory the Allrun script can be run.

```
./Allrun
```

3.7 Analysis and Results

One of the features of the Rayleigh-Bénard Convection flow is that the area (and time) averaged Nusselt number in the cube given by Equation (3.1) is constant in the y-axes once a steady state has been reached. Further, the volume (and time) averaged Nusselt number, hereon referred to as global Nusselt Number, calculated from Equation (3.1) should also be equal to this value. To assess this we took slices at 5 different heights in the cell; in the middle, at one boundary layer height from the horizontal walls and at twice the boundary layer height from the horizontal walls. By looking in the surfaceFieldValue.dat files of the relevant slices we can assess whether the above is true; an example command is given by

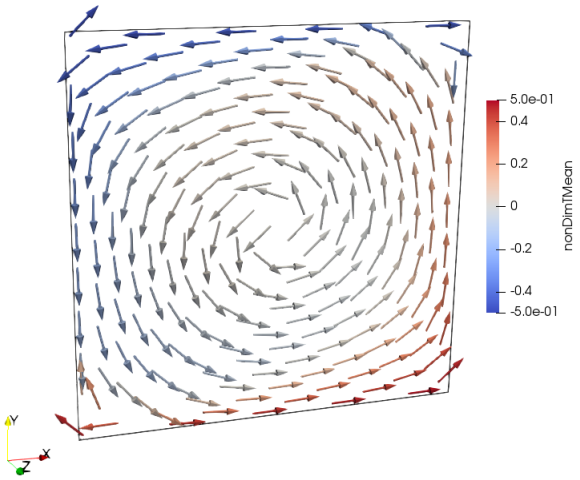
```
tail -1 postProcessing/areaAveragePlaneVeryBottom/0/surfaceFieldValue.dat
```

Slice	$\mathrm{Nu}_{\mathrm{comp1}}$	$\mathrm{Nu}_{\mathrm{comp2}}$	$\mathrm{Nu}_{\mathrm{tot}}$
Very top	1.87	-2.06	3.93
Top	3.61	-0.23	3.84
Middle	3.85	0.03	3.82
Bottom	3.59	-0.24	3.84
Very Bottom	1.83	-2.09	3.92
Global	2.86	-1	3.86

Table 3.1: Nu summary

This is not a validation exercise for the solver, the purpose of the tutorial is to show proof of concept, as such we can only make observations with the caveat that more work needs to be done for a fully validated solver. However, the Nusselt summary table suggests that the solution has reached a statistically steady state as the total Nu is almost constant up and down the square domain.

One further observation is the different contributions to the total Nusselt number from the two components. It can be seen that in the bulk of the flow the *convective* component 1 is dominant and the reverse is true as we approach the boundary layers where the *diffusive* component 2 takes over. A further qualitative observation is the typical nature of the time-averaged LSC velocity vector field which is observed by looking at the nonDimUMean field from the final time-step (in paraview use filters → cell-centres then filters → glyph).



Chapter 4

Study questions

1. What are the main differences between *buoyantPimpleFoam* and the low-Mach number equivilant explained in this document ?
2. How is continuity enforced in the transient variable density low-Mach-number PISO algorithm?
3. How can the algorithm presented be further devoloped so that the temperature (and hence density) correction step is included in the transient PISO algorithm ?
4. What function object can we use to calculate the Nusselt number at the hot and cold walls during the simulation ?

It is noted here that the function objects outlined in this tutorial are particularly useful for turbulent (hence chaotic) simulations which do not tend to a steady state. As such the advanced user is encouraged to increase the Rayleigh number and to model the same case in 3D.

Chapter 5

Hints and tricks

During the course of creating this tutorials many problems have been encountered. The following are a summary of tips and tricks that may help the user should they carry out a similar task.

1. More than just adding robustness, the hydrodynamic pressure field is essential for this solver. In their reactingLMFoam solver, Nogenmyr et Al [7] treat buoyancy as a volumetric source term in the momentum equation and this produces no velocity field when combined with the other changes outlined here.

2. The pressure used in the constrainHbyA function in the pEqn.H must be the thermodynamic pressure, which is not immediately obvious from the *buoyantPimpleFoam* code.

3. The implicit addition and explicit extraction of $\frac{\partial(\psi\, p_d)}{\partial t}$ in the pressure poisson equation [8] is **necessary** but not always sufficient to stablize the solution of the pressure Poisson equation. The tolerances and relTol may need to be reduced in fvSolution.

4. The equationOfState used in the thermophysicalProperties dictionary plays an important role. With this solver it is suggested to use the perfectGas even when p^0 is equal to p_{ref} (i.e. even in an open domain). This is because the use of incompressiblePerfectGas equation of state does correctly set p^0 to p_{ref}, however it also sets ψ, and consequently the ddt(psi)*p term, to zero in the the PISO loop.

Bibliography

[1] G. Ahlers, S. Grossman, and D. Lohse. "Heat transfer and large scale dynamics in turbulent Rayleigh Benard convection". In: *Rev. Mod. Phys* 81.00 (2009).

[2] Ferziger J. and Peric H. *Computational Methods for Fluid Dynamics*. Springer-Verlag Berlin Heidelberg, 2002.

[3] P.J. Oliviera and R.I. Issa. "An improved PISO algorithm for the computation of buoyancy driven flows". In: *Numer. Heat Tr. Bfund* 40.473-493 (2001).

[4] B. Lessani and M.V. Papalexandris. "Time accurate calculation of variable density flows with strong temperature gradients and combustion". In: *J. Comput. Phys.* 212.218-246 (2006).

[5] H. Jasak. "Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows". PhD thesis. Imperial College, 1996.

[6] I. Demirdzic, M. Peric, and Z Lilek. "A colocated finite volume method for predicting flows at all speeds". In: *Int. J. Num. Meth. Fluids* 16.1029-1050 (1993).

[7] Nogenmyr, K.J. and Chan, C.K. and Duwig, C. *Finite rate chemistry effects and combustor liner heat transfer studies in a framework of LES of turbulent flames*. `http://web.student.chalmers.se/groups/ofw5/Presentations/KarlJohanNogenmyrSlidesOFW5.pdf`. Accessed: 27-11-2018.

[8] Souvandy-Tabarracci, D. and Lamorlette, A. and Morvan, D. *Modele de propagation d'un feu de foret en low Mach number*. `http://docs.gdrfeux.univ-lorraine.fr/Balma/M2P2_1.pdf`. Accessed: 27-11-2018.

[9] O. Shishkina and C. Wagner. "Local heat fluxes in turbulent Rayleigh Benard convection". In: *Phys. Fluids* 19.085107 (2009).

[10] R. Stevens, R. Verzicco, and D. Lohse. "Radial boundary layer structure and Nusselt number in Rayleigh Benard convection". In: *J. Fluid Mech.* 643.495-507 (2010).

[11] O. Shiskina et al. "Boundary layer structure in turbulent thermal convection and its consequences for the required numerical resolution". In: *New J. Phys.* 12.07502 (2010).

[12] R.I. Issa. "Solution of the Implicitly Discretized Fluid Flow Equations by Operator-Splitting". In: *J. Comp. Phys.* 62.40-65 (1985).

[13] Churchfield, M. *BuoyantBoussinesqPisoFoam*. `https://openfoamwiki.net/index.php/BuoyantBoussinesqPisoFoam`. Accessed: 27-11-2018.